По первым символам файла «<?php» можно понять, что это исполняемый скрипт, написанный на php.

В последних строчках представлен код, в котором переменные $FHC, $XfL, $bqcl, $DSG и $LAm инициализируются через битовую операцию XOR (^) между строками.

С помощью следующего скрипта python можно получить значения этих переменных:

```
import re

# Имена переменных, которые нужно извлечь
target_vars = {'DSG', 'FHC', 'LAm', 'XfL', 'bqcl'}

# Словари для хранения зашифрованных значений и ключей
encrypted_values = {}
keys = {}

# Чтение файла с именем "1"
with open('1', 'r', encoding='utf-8', errors='ignore') as f:
    content = f.read()

# Регулярное выражение для поиска строк вида $var='...'^"...";
pattern = r'\$(\w+)\s*=\s*\'([^\']*)\'\s*\^\s*"([^"]*)"\s*;'

matches = re.findall(pattern, content)

for var_name, encrypted_str, key in matches:
    if var_name in target_vars:
        # Преобразуем экранированные символы (например, \n, \t и т.д.) в настоящие байты/символы
        # Но в данном случае encrypted_str — это строка с буквальными символами (возможно, недру́желюбными для печати)
        # Поэтому просто сохраняем как есть (в виде строки, где каждый символ — это байт)
        encrypted_values[var_name] = encrypted_str
        keys[var_name] = key

# Проверка: все ли переменные найдены
missing = target_vars - set(encrypted_values.keys())
if missing:
    raise ValueError(f"Не найдены переменные в файле: {missing}")

# Функция для расшифровки через XOR
def xor_decrypt(encrypted_text, key):
    # Расширяем ключ до длины зашифрованного текста
    extended_key = (key * ((len(encrypted_text) // len(key)) + 1))[:len(encrypted_text)]
    decrypted = ''.join(chr(ord(c) ^ ord(k)) for c, k in zip(encrypted_text, extended_key))
    return decrypted

# Расшифровываем
decrypted = {}
for var in target_vars:
    decrypted[var] = xor_decrypt(encrypted_values[var], keys[var])

# Вывод результатов
for var in ['DSG', 'FHC', 'LAm', 'XfL', 'bqcl']:
    print(f"{var}:", repr(decrypted[var]))  # используем repr, чтобы видеть непечатаемые символы
```

Полученные значения:
DSG: 'preg_match'
FHC: 'create_function'
LAm: 'file'
XfL: 'gzinflase'
bqcl: 'implode'

Функция create_function представляет выполнение кода
gzinflase соответствует функции gzinflate, которая выполняет распаковку сжатых данных.
implode — функция для объединения строк.
preg_match — регулярное выражение для извлечения данных.
file — чтение содержимого файла.

С помощью замены переменных из исходного файла получим следующую функцию:

```
<?php
/** data **/
@("create_function")(
    "",
    "};" .
        ("gzinflate")(
            ("preg_match")(
                "#/\*\*(.*)\*\*/#s",
                ("implode")("", ("file")(__FILE__)),
                $match
            )
                ? $match[1]
                : ""
        ) .
        "//"
);
```

Итак, код работает следующим образом:
Читает текущий PHP-файл.
Извлекает все данные, заключенные между /** и **/.
Распаковывает полученные данные функцией **gzinflate**.
Выполняет полученный и распакованный код.

Снова навайбкодим скрипт для извлечения данных из комментария /** raw_data **/

```
import re
import base64

# Имя файла с вредоносным PHP-кодом
filename = '1'  # или 'malware.php'

# Читаем файл в бинарном режиме
with open(filename, 'rb') as f:
    content = f.read()

# Ищем шаблон: /** ... **/
# Используем re.DOTALL, чтобы . совпадал с \n
```

```
match = re.search(rb'/\*\*(.*?)\*\*/', content, re.DOTALL)

if match:
    raw_data = match.group(1)  # это bytes!
    b64_encoded = base64.b64encode(raw_data).decode('ascii')
    print("Base64-encoded payload:")
    print(b64_encoded)

    # Опционально: сохранить в файл
    with open('payload.b64', 'w') as out:
        out.write(b64_encoded)
    print("\nСохранено в payload.b64")
else:
    print("Не найдено содержимое между /** и **/")
```

Для просмотра исходного вредоносного кода можно использовать онлайн песочницу (например https://onlinephp.io/).

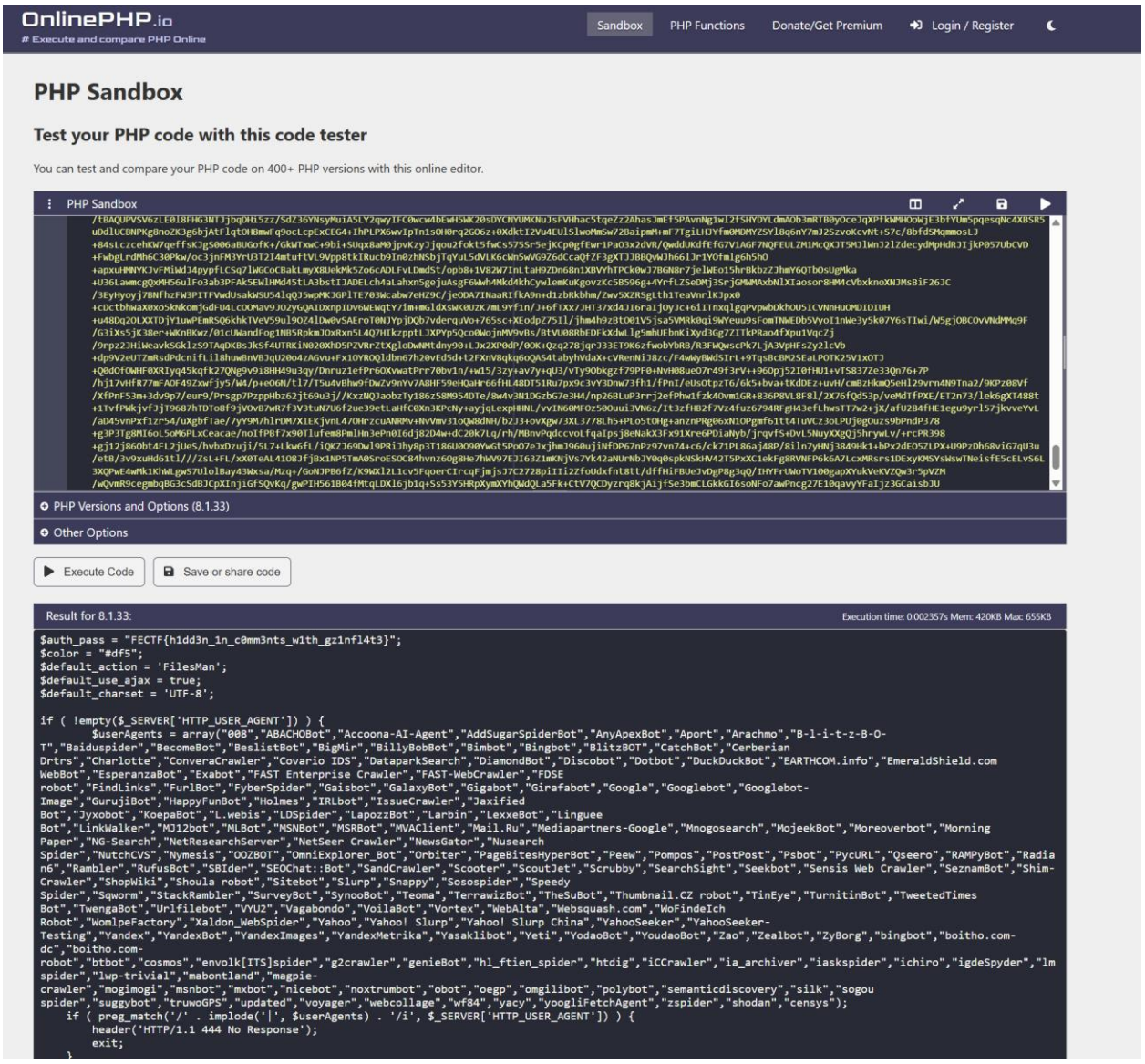
Выполним следующий код:

<?php

print(gzinflate(base64_decode('')));

Где в base64_decode('') подаются данные, расположенные между /** и **/ из исходного файла **в кодировке base64**.

Получаем исходные код скрипта:



ФЛАГ: FECTF{h1dd3n_1n_c0mm3nts_w1th_gz1nfl4t3}

Для создания задания переупаковывал так:

```
<?php

// Исходная base64-строка (замените на реальную)
$base64_encoded = 'eJxLtDK0NDI2MTUzt7C0sraxtXNwdHJ2cXVz9/D08vbx9fMP...'; // ← ВСТАВЬТЕ СЮДА НАСТОЯЩУЮ СТРОКУ

// 1. Декодируем base64
$compressed = base64_decode($base64_encoded);

if ($compressed === false) {
    die("Ошибка: не удалось декодировать base64.\n");
}

// 2. Распаковываем с помощью gzinflate
$uncompressed = @gzinflate($compressed);

if ($uncompressed === false) {
    // Попробуем gzuncompress, если gzinflate не сработал
    $uncompressed = @gzuncompress($compressed);
    if ($uncompressed === false) {
        die("Ошибка: не удалось распаковать данные (проверьте, что строка действительно сжата через gzdeflate/gzcompress).\n");
    }
}

// 3. Заменяем хеш на флаг
$old_hash = '9da7ce1c7964988ebbb842d61703eb9f97c734bc';
$new_flag = 'FECTF{h1dd3n_1n_c0mm3nts_w1th_gz1nfl4t3}';

$modified_text = str_replace($old_hash, $new_flag, $uncompressed);

// Убедимся, что замена произошла
if ($modified_text === $uncompressed) {
    echo "Внимание: хеш '$old_hash' не найден в распакованном тексте.\n";
}

// 4. Сжимаем обратно (используем gzdeflate — это то, что ожидает gzinflate)
$recompressed = gzdeflate($modified_text);

if ($recompressed === false) {
    die("Ошибка: не удалось сжать данные.\n");
}

// 5. Кодируем в base64
$new_base64 = base64_encode($recompressed);

// Выводим результат
echo "Новая base64-строка:\n";
echo $new_base64 . "\n";

// (Опционально) Проверка: распакуем и выведем текст для подтверждения
$test = gzinflate(base64_decode($new_base64));
echo "\nПроверка (распакованный текст):\n";
echo $test . "\n";

?>
```

Далее делаем из base64 текст и обновляем скрипт

```
import base64


# Входная base64-строка

filename = "payload.b64"

with open(filename, 'r') as f:

    base64_recode = f.read()


# Декодируем из base64 → получаем байты

decoded_bytes = base64.b64decode(base64_recode)
```

```
# Сохраняем как текстовый файл
with open('raw.php', 'wb') as out:
    out.write(decoded_bytes)

print("Сохранено в raw.php")
```

Дальше я уже как-то руками перенес начало и конец скрипта (при этом кодируя в base64 при переносе и декодирую обратно, чтобы не потерять hex значения при изменении кодировки). Наверно можно было умнее сделать, но мне лень.